Distributed Systems in Action

Mohamed Sweelam

Software Engineer
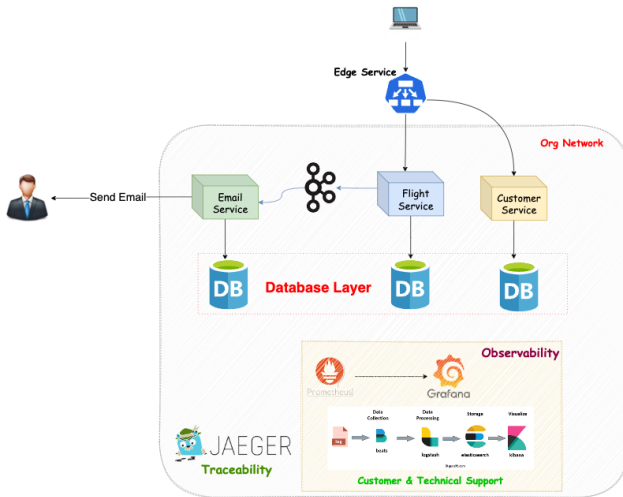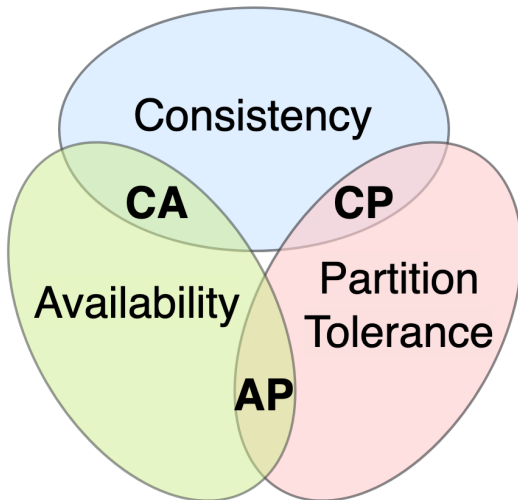
Outline

Introduction to Distributed Systems

- Definition and characteristics of distributed systems
- Importance and benefits of distributed systems
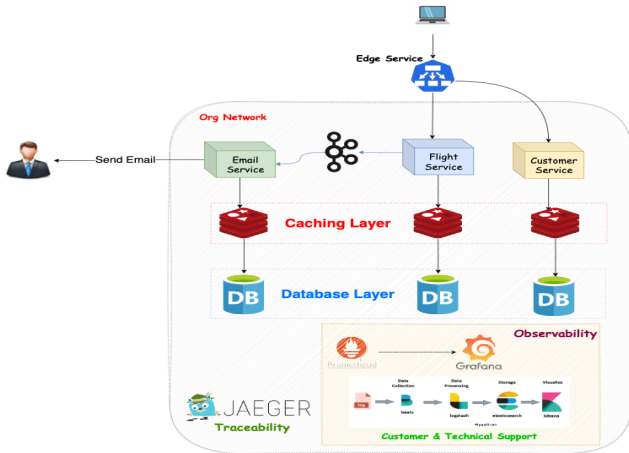- Challenges and trade-offs in designing distributed systems

Use Case: Flight System

CAP Theorem

High Availability and Scalability

- Load Balancer
- API Gateway
- Rate Limiter

Resiliency

```yaml
spring:
  cloud:
    default-filters:
      - name: Retry
        args:
          retry: 5
          methods: GET
          backoff:
            firstBackOff: 10ms
            maxBackOff: 50ms
            factor: 2
            basedOnPreviousValue: false
```
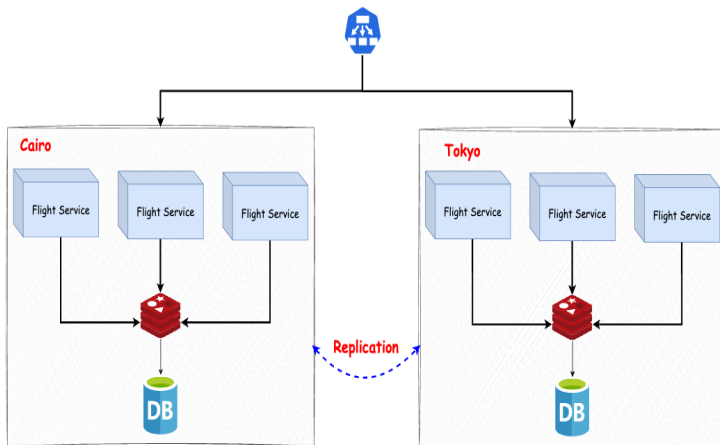
Data Replication

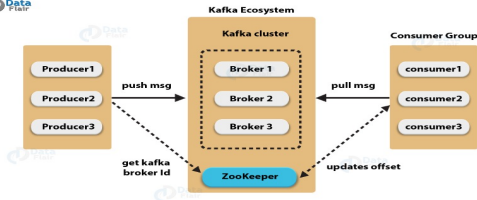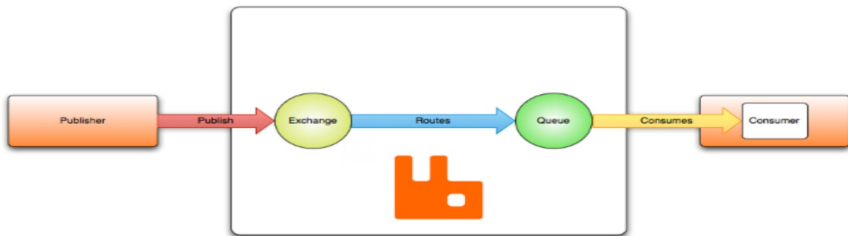# Distributed Messaging Systems

- Introduction to distributed messaging systems
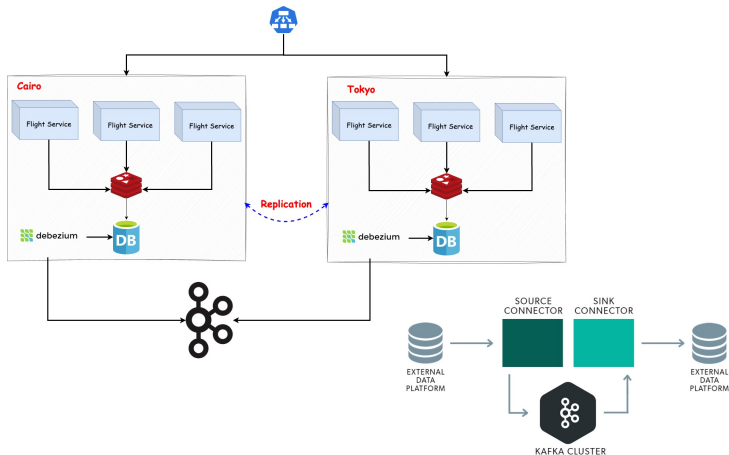- Overview of Kafka and RabbitMQ

RabbitMQ vs Kafka

"Hello, world" example routing

Data Replication: CDC

In databases, **change data capture (CDC)** is a set of *software design patterns* used to determine and track the data that has changed (the "deltas") so that action can be taken using the changed data.

Data Replication: CDC

```json
{
  "schema": {},
  "payload": {
    "source": {
      "version": "2.2.1.Final",
      "connector": "mysql",
      "name": "mysql",
      "db": "inventory",
      "sequence": null,
      "table": "customers"
    },
    "databaseName": "inventory",
    "schemaName": null,
    "ddl": "ALTER TABLE customers ADD middle_name varchar(255) AFTER first_name",
    "tableChanges": [
      {
        "type": "ALTER",
        "id": "\"inventory\".\"customers\"",
        "table": {
          "defaultCharsetName": "utf8mb4",
          "primaryKeyColumnNames": [
            "id"
          ],
          "columns": [
            {
              "name": "id",
              "jdbcType": 4,
              "nativeType": null,
              "typeName": "INT",
              "typeExpression": "INT",
              "charsetName": null,
              "Length": null,
              "scale": null,
              "position": 1,
              "optional": false,
              "autoIncremented": true,
              "generated": true
            }
          ]
        }
      }
    ]
  }
}
```

Distributed Messaging Systems

- Introduction to distributed messaging systems
- Overview of Kafka and RabbitMQ
- Comparison between Kafka and RabbitMQ:

Kafka	RabbitMQ
High-throughput, fault-tolerant distributed streaming platform.	Robust and flexible messaging broker.
Emphasizes real-time event streaming and data pipeline use cases.	Implements Advanced Message Queuing Protocol (AMQP).
Provides strong durability and replication guarantees.	Focuses on message queuing and asynchronous communication.
Scales horizontally to handle large-scale data streams.	Provides various messaging patterns (e.g., publish-subscribe, point-to-point).
Supports complex event processing with built-in stream processing.	Offers pluggable message durability, routing, and acknowledgement mechanisms.

Distributed Key-Value Stores

- Introduction to distributed key-value stores (e.g., Dynamo, Cassandra)
- Data partitioning and replication techniques
- Consistency and availability trade-offs

Distributed System Security

- Security challenges in distributed systems
- Authentication and access control in distributed environments
- Distributed denial-of-service (DDoS) prevention techniques

Distributed System Monitoring

- Log aggregation and distributed tracing
- Metrics collection and monitoring tools (e.g., Prometheus, Grafana)
- Anomaly detection and performance optimization

API Documentations
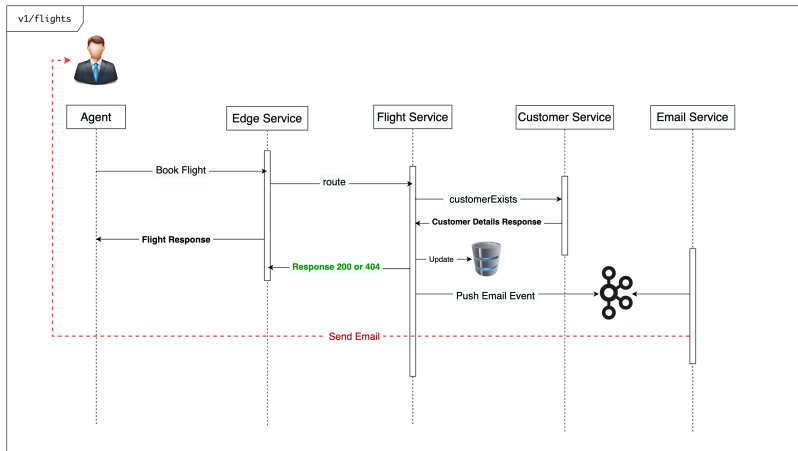
Conclusion

- Recap of distributed systems concepts
- Overview of various distributed systems topics
- Further resources for exploring distributed systems in depth